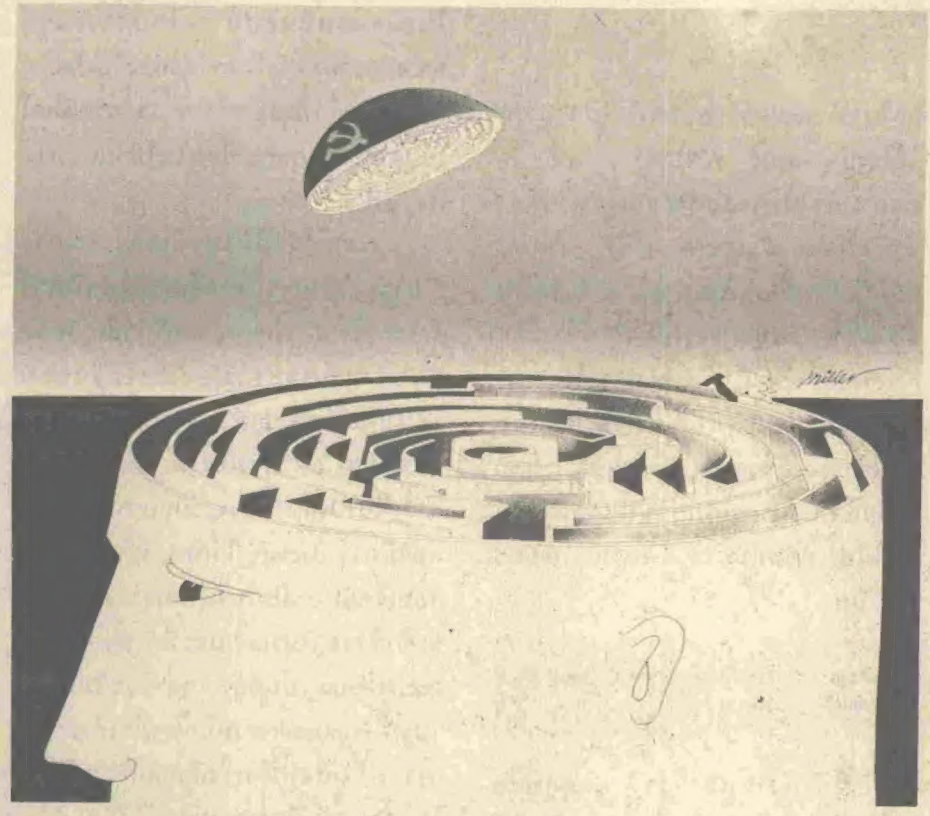
เรื่องน่าอ่าน
สุรวุฒิ สุชินโรจน์

สถาปัตยกรรม ระบบผู้เชี่ยวชาญ (จบ)



ติดตามการเขียนความรู้ในระบบผู้เชี่ยวชาญและกฎเกณฑ์ที่นำไปสู่แสงสว่างของระบบผู้เชี่ยวชาญ

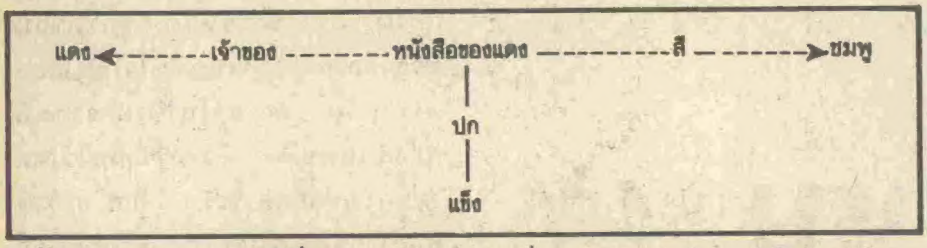
ท่านผู้อ่านคงได้ทำความเข้าใจถึงความหมายที่มา และแบบการสร้าง "ส่วนแกน" ของโปรแกรมระบบผู้เชี่ยวชาญพอเป็นสังเขปแล้ว คราวนี้ผู้เขียนจะขอกล่าวถึงเรื่องราวของการเขียนความรู้ในรูปกฎเกณฑ์และกลไกของการใช้ปัญญาในโปรแกรมนี้ เพื่อให้ได้เข้าใจอย่างถ่องแท้มากยิ่งขึ้น

การเขียนความรู้ในโปรแกรม

เทคนิคสำคัญประการหนึ่งในโปรแกรมระบบผู้เชี่ยวชาญ คือการเขียนหรือแสดงความรู้หลายชนิดไว้ในรูปแบบต่าง ๆ กัน ความรู้ในโปรแกรมจะมีสองลักษณะคือ ความรู้ที่เก็บไว้โดยไม่มีการเปลี่ยนค่าไปมา ได้แก่ ความรู้พวกรายชื่อ วิธีการคำนวณ กฎเกณฑ์ต่าง ๆ เรียกว่าเป็นความรู้สถิต (static knowledge) ต่างกับความรู้อีกประเภทหนึ่ง ซึ่งเป็นความรู้ที่ได้มาจากการอนุมานของโปรแกรมหรือการถามจากผู้ใช้เรียกว่า "ความรู้พลวัต" (dynamic knowledge) เนื่องจากเปลี่ยนแปลงได้เสมอ

ความรู้ทั้งสองลักษณะนี้มีวิธีการแสดงได้หลายแบบในโปรแกรมคอมพิวเตอร์ เช่น โดยการใช้เน็ตเวอร์ก เช่น semantic net หรือ petri net ใช้เพื่อเน้นความสัมพันธ์กันระหว่างหน่วยของความรู้ หรือแสดงในรูปต้นไม้ (tree) เพื่อเน้นความเป็นลำดับชั้นสูงต่ำ เช่น ความรู้เกี่ยวกับลำดับญาติพี่น้องหรือแสดงในรูปกรอบ (frame) เน้นความเป็นกลุ่มหรือเป็นหน่วยของความรู้และแสดงในรูปสคริปต์ (script) เมื่อสนใจที่ลำดับของเหตุการณ์หรืออาจใช้รวม ๆ กันหลายรูปแบบในบางโปรแกรม (รายละเอียดของการศึกษาเกี่ยวกับการแสดงความรู้ในโปรแกรมเป็นการศึกษาในหัวข้อ knowledge representation อาจหาอ่านได้จาก (RICH) ขอยกตัวอย่างการแสดงส่วนของความรู้ "หนังสือของแดง" ด้วย semantic net ดังรูปที่ 1 ซึ่งสามารถเขียนเป็นภาษา LISP โดยให้ DANG-BOOK เป็นอะตอมมี property list เป็น (owner DANG) (color PINK) (cover HARD-COVER)

รูปแบบการแสดงความรู้ในความจำชั่วคราว (ฐานข้อมูล) ที่ใช้กันมากในโปรแกรมระบบผู้เชี่ยวชาญทั่ว ๆ ไป



รูปที่ 1 Semantic Net แสดงความรู้เกี่ยวกับ "หนังสือของแดง" ซึ่งบอกว่า "แดงเป็นเจ้าของหนังสือ" "หนังสือของแดงมีสีชมพู" และ "หนังสือของแดงเป็นหนังสือปกแข็ง"

เป็นการเก็บในลักษณะที่เป็นกรณีพิเศษของ semantic net อย่างหนึ่ง ซึ่งเรียกกันว่าแบบ attribute-object-value triplet (A-O-V) คือจัดไว้ในฟอร์มที่เรียงลำดับเป็น (คุณสมบัติ สิ่ง ค่า) เช่น (สี ลูกบอล แดง) เพื่อแทนลูกบอลสีแดง โครงสร้างของฟอร์มอาจมีความซับซ้อนได้โดยการกำหนดให้ส่วนของ "คุณสมบัติ" เป็นโครงสร้างข้อมูล (data structure) แทนที่จะเป็นอะตอมคือหน่วยของข้อมูลที่แบ่งแยกไม่ได้ ขอยกตัวอย่างการใช้ A-O-V มีฟอร์มดังรูปที่ 2

```
แบบฟอร์ม : (หัวข้อ พารามิเตอร์ ค่า)
ตัวอย่าง  : (เชื้อ ชนิด เกลียว)
```

รูปที่ 2 ตัวอย่างการใช้ A-O-V

"ชนิด" มีลักษณะประกอบดังนี้

1. คาดว่าเป็น : (หนึ่งใน) (รายชื่อเชื้อ)
2. มีข้อมูลจากแล็บหรือไม่ : มี

ในที่นี้ "รายชื่อเชื้อ" เป็นโครงสร้างข้อมูลที่เป็น list และ "หนึ่งใน" เป็นฟังก์ชัน

ความรู้ในรูปกฎเกณฑ์

ดังได้กล่าวมาแล้วว่าคำสั่งสำคัญที่สุดในโปรแกรมระบบผู้เชี่ยวชาญคือคำสั่งที่เป็นความรู้ในรูปกฎเกณฑ์แสดงอยู่ในรูป implication โดยทั่วไปสามารถใช้โอเปอเรเตอร์ AND/OR/NOT ได้ทั้งสามตัวในการเขียนส่วนเงื่อนไขของคำสั่ง เช่น a and (b or c) —> ... อย่างไรก็ตาม a หรือ b หรือ c ไม่จำเป็นต้องเป็นอะตอมอาจเป็นฟังก์ชันเป็น predicate หรือเป็น statement ในภาษาที่ใช้เขียนโปรแกรมอาจมีส่วนประกอบสลับซับซ้อนในทางปฏิบัติ expert system shell แต่ละภาษามักมีส่วนนี้แตกต่างกันออกไป ตัวอย่าง a ในภาษา OPSS อาจเป็น

```
(Person ^name <someone> ^age
{<child> <15})
```

ซึ่งมีความหมายว่า บุคคลที่ชื่ออะไรก็ได้ ซึ่งมีอายุน้อยกว่า 15 ในที่นี้ Person เป็นออบเจ็คต์ที่มีแอตทริบิวต์คือ ^name และ ^age (ใส่เครื่องหมาย caret เพื่อให้ต่างออกไป) ส่วนที่อยู่ใน <...> เป็นตัวแปรที่สามารถแทนค่าได้ด้วยสิ่งที่มีแอตทริบิวต์ชนิดเดียวกัน การแทนค่านี้เป็นวิธีการที่เรียกว่า instantiation คือแทนค่าด้วยค่าเดียวกันกับตัวแปรตัวเดียวกันที่ปรากฏอยู่ในคำสั่งในส่วนอื่น ๆ ทุกส่วน เช่น ถ้ามีตัวแปรเดียวกันปรากฏในคำสั่งนี้สองที่ก็แทนทั้งสองที่ด้วยค่าเดียวกัน ตัวอย่างของเงื่อนไขข้างบนนี้จะเป็นจริงกับข้อมูลในความจำชั่วคราวดังนี้

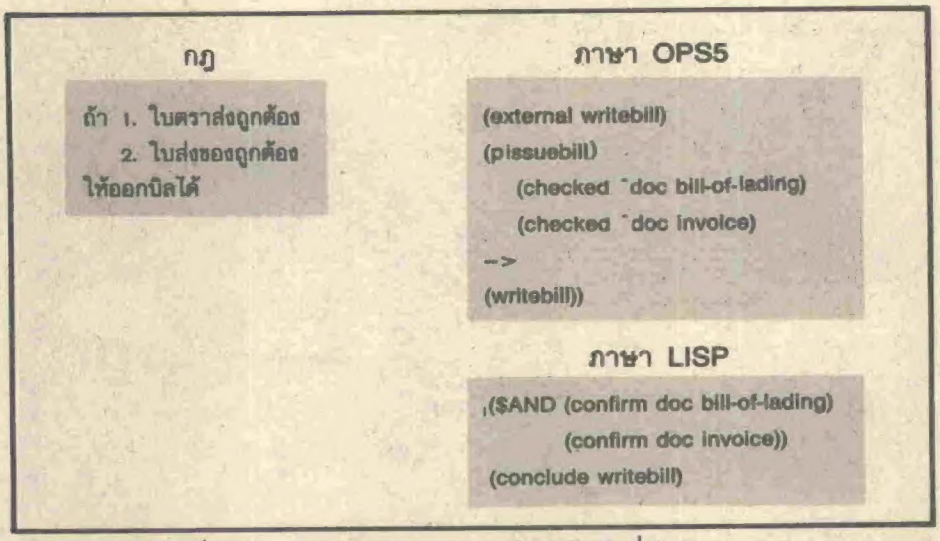
```
(Person ^name Steve ^age 14)
แปลว่า มีคนชื่อ Steve อายุ 14 ปี
(Person ^name Mary Smith ^age
12) แปลว่า มีคนชื่อ Mary Smith อายุ 12 ปี
```

จะยกตัวอย่างกฎเต็ม ๆ ทั้งข้อให้ดู สมมติว่าในบริษัทของเรามีหลักข้อหนึ่งในการออกบิลลูกค้า โดยการตรวจความถูกต้องของใบตราส่งและใบส่งของก่อนทุกครั้งที่จะออกบิล เราอาจเขียนเป็นรูปของกฎและเขียนในโปรแกรมได้ดังรูปที่ 3

ความรู้ที่เป็นโปรซีเดอร์จะมีการเก็บไว้ในโปรแกรมด้วยชื่อของโปรซีเดอร์ ซึ่งจะเหมือนกับค่าของแอตทริบิวต์บางชนิด ตัวของรูทีนจะรวมอยู่กับโปรแกรมระบบผู้เชี่ยวชาญส่วนที่เป็นกลไกของโปรแกรม แต่ไม่อยู่ในโครงสร้างข้อมูลใด ๆ ในโปรแกรมบางระบบความรู้เหล่านี้ต้องมีการ declare ไว้ก่อน เช่นเดียวกับออบเจ็คต์ทุกชนิดในโปรแกรม ดังนั้นในชุดคำสั่งของโปรแกรมจะมีคำสั่งประเภท declaration ปนอยู่ด้วยอาจจะไม่เขียนในรูป implication เหมือนอย่างคำสั่งอื่น เพราะเป็นสิ่งที่ทำครั้งเดียวแล้วไม่ได้ใช้ในโปรแกรมอีกเลย

กลไกของการใช้ปัญญา

ความรู้ต่าง ๆ ที่แสดงไว้ในโปรแกรมจะถูกนำมาใช้ในการแก้ปัญหา ซึ่งมีกลไกหลักได้แก่ การตั้งสมมติฐาน การพิสูจน์สมมติฐาน และการเลือกระหว่างคำตอบหลาย ๆ คำตอบ การตั้งสมมติฐานคือการตั้งเป้าหมายของการอนุมานขึ้นมาหลาย ๆ เป้าหมายโปรแกรมแต่ละโปรแกรมอาจจะมีเป้าหมายเหล่านี้ตั้งไว้ตายตัวก่อนแล้วก็ได้ หรือจะเสนอขึ้นมาโดยการอนุมานจากข้อมูลพื้นฐานก็ได้ หลังจากนั้นโปรแกรมก็พยายามจะ



รูปที่ 3 ทางซ้ายคือกฎที่เราต้องการจะใช้ในโปรแกรมทางขวาคือกฎเดียวกันที่เขียนในภาษา OPS5 และภาษา LISP

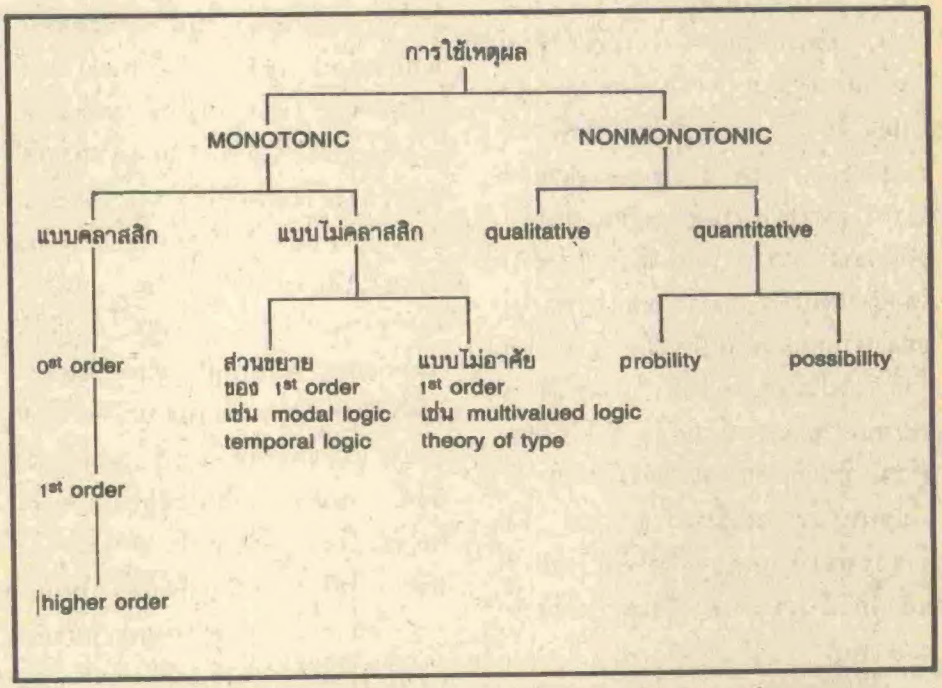
สืบสาวโดยใช้ความรู้ที่เป็นกฎเกณฑ์ประกอบกับหลักฐานทำการอนุมานเพื่อให้ได้ข้อสรุปมาเปรียบเทียบว่าสมมติฐานใดมีความน่าจะเป็นไปได้ที่สุด เลือกเอาสมมติฐานนั้นเป็นคำตอบทั้งสามขั้นตอนนี้อาศัยกลไกหลักคือการอนุมาน

การอนุมานเป็นไปตามหลักการใช้เหตุผลซึ่งมีอยู่หลายวิธี จะเขียนเป็นแผนผังให้ดูเปรียบเทียบกับชนิดของตรรกะ (logic) เพื่อจะได้เห็นตำแหน่งที่มาของความคิดในบรรดาวิธีการทั้งหลายทางตรรกะโดยกว้าง ดังรูปที่ 4

แผนผังที่แสดงให้ดูแบ่งตรรกะเป็นสองกลุ่มคือ กลุ่มที่ "ความจริงในระบบ" ไม่มีการลดถอยลงพวกนี้เรียกว่า monotonic logic ประกอบด้วยตรรกะแบบมาตรฐานทั้งที่เป็นคลาสสิกและที่พัฒนาแยกออกไป ซึ่งได้รับการศึกษาไว้อย่างดีที่สุดโดยนักตรรกศาสตร์จำนวนมาก วิธีการที่พัฒนาไว้นับว่าก้าวหน้ากว่า nonmonotonic logic มาก ตัวอย่างเช่นการใช้ resolution algorithm สำหรับการพิสูจน์ทฤษฎีบท (theorem) ของ 1st order logic หรือที่เรียกว่า predicate calculus อย่างที่ใช้ในโปรแกรมพวกพิสูจน์ทฤษฎีบท (theorem prover) แต่เป็นที่น่าเสียดายว่ามีปัญหาบางอย่างที่ทำให้เราไม่สามารถใช้ประโยชน์วิธีการทางตรรกศาสตร์แบบมาตรฐานในโปรแกรมระบบผู้เชี่ยวชาญโดยตรง สืบเนื่องจากสาเหตุเหล่านี้คือ

1. ตรรกะแบบ monotonic ไม่มีกลไกที่จะจัดการกับการอนุมานในสถานการณ์ที่ข้อมูลหรือหลักฐานไม่พร้อมและไม่สามารถบอกได้ว่าจริงหรือไม่ เพราะหลักฐานบ่งความจริงที่ขัดแย้งกันเอง อันเป็นสถานการณ์การอนุมานของผู้เชี่ยวชาญมนุษย์ แม้ว่าจะมีทางเพิ่มกลไกบางอย่างให้กับตรรกะแบบนี้เพื่อให้รับสถานการณ์ดังกล่าว ซึ่งในปัจจุบันแนวทางนี้ยังไม่มีการพัฒนา

2. วิธีการที่ใช้ในโปรแกรมพิสูจน์ทฤษฎีบทส่วนใหญ่ใช้วิธีการที่ปรับปรุงมาจากวิธีของ Herbrand (ซึ่งแสดงไว้ตั้งแต่ปี 1930) ตัวอย่างเช่น resolution algorithm ของ J.A. Robinson วิธีการเหล่านี้แม้ว่าเป็นอัลกอริทึมที่มีประสิทธิภาพ (การคำนวณเป็น polynomial time ในบางกรณี) แต่เป็นวิธีการพิสูจน์



รูปที่ 4 แผนผังแสดงตรรกะชนิดต่าง ๆ เปรียบเทียบระหว่างตรรกะที่ใช้ในโปรแกรมระบบผู้เชี่ยวชาญกับตรรกะแบบทั่วไป ตรรกะของโปรแกรมระบบผู้เชี่ยวชาญมักเป็นแบบ NONMONOTONIC

ข้อความที่เป็นจริงเท่านั้นจึงใช้ไม่ได้กับข้อความที่ไม่จริง เนื่องจากไม่มีทางที่จะคิดอัลกอริทึมสำหรับพิสูจน์ว่าข้อความเป็นจริงหรือไม่ สำหรับ 1st order logic ความจริงข้อนี้เป็นบทพิสูจน์ของ Church (ท่านผู้นี้เป็นผู้เสนอวิธีการคำนวณอย่างหนึ่งที่นิยมใช้เป็นโมเดลทางคณิตศาสตร์ของการโปรแกรมในปัจจุบันเรียกว่า แลมป์ดาแคลคูลัส) ด้วยเหตุนี้ถึงแม้จะเพิ่มกลไกของตรรกะให้สามารถทำงานกับสถานการณ์ที่ต้องการได้ ระบบตรรกะใหม่อย่างน้อยก็จะเท่าเทียม 1st order logic ดังนั้นการพิสูจน์ข้อความในตรรกะระบบใหม่จะเผชิญกับปัญหาเดียวกัน

ตรรกศาสตร์แบบ nonmonotonic ยอมรับสถานการณ์การอนุมานในลักษณะที่ต้องใช้ในโปรแกรมระบบผู้เชี่ยวชาญ กล่าวคือ สามารถสรุปข้อเท็จจริงด้วยปริมาณหลักฐานเท่าที่มีเมื่อมีหลักฐานเพิ่มขึ้น หรือเปลี่ยนไปเป็นอื่นก็สามารถปรับข้อสรุปนั้นได้ อย่างไรก็ดี ตรรกะแบบ nonmonotonic ไม่มีวิธีสำหรับการพิสูจน์ความจริงของข้อความของตัวเอง คือไม่มี axiom จึงไม่มีการพิสูจน์ consistency หรือ completeness ของระบบ ทำให้เราไม่สามารถระบุความถูกต้องของการทำงานของตรรกะเหล่านี้ได้อย่างสมบูรณ์ เราจึงเชื่อผลการทำงานของโปรแกรมระบบผู้เชี่ยวชาญได้ โดยการนำไปใช้จริงเท่านั้น ถ้าใช้ได้ผลระบบก็ "อาจ" จะถูกต้อง ถ้าไม่ได้ผลระบบก็มีข้อผิดพลาดบางอย่างในลักษณะเดียวกับการจ้างผู้เชี่ยวชาญมนุษย์ ซึ่งอาจจะทำงานถูกต้องก็ได้หรืออาจผิดพลาดก็ได้ เราไม่มีทางรู้จนกว่าผลงานจะปรากฏ

วิธีทางตรรกะแบบ nonmonotonic มีหลายแนวทางแบ่งกว้าง ๆ ได้สองแนวทางคือ แบบที่พยายามสร้างโมเดลทางคณิตศาสตร์เป็นพื้นฐาน กับแบบที่พยายามสร้างเป็นกลไกมักจะเลียนวิธีการใช้เหตุผลที่เข้าใจกันว่ามนุษย์ใช้อยู่ในชีวิตประจำวัน แบบแรกนั้นมีสองพวก พวกหนึ่งถือความหมายของค่าความเป็นไปได้ (probability) เป็นสำคัญ เนื่องจากมีทฤษฎีความเป็นไปได้ที่พัฒนาไว้ดีแล้วได้แก่ วิธีการของเบย์ (Bayesian approach) ซึ่งเว้นแต่โปรแกรมวิเคราะห์ข้อมูลทางการแพทย์แล้วไม่นิยมใช้ที่อื่นอีก พวกที่สองไม่ถือความหมายของค่าความเป็นไปได้เป็นสำคัญ แต่ตั้งค่าขึ้นมาใหม่เป็นค่า "อาจเป็นไปได้" (possibility) ซึ่งเป็นค่าที่เพิ่มหรือลดลงได้ขึ้นกับหลักฐานที่รวบรวมมาได้ โดยเฉพาะไม่มีข้อจำกัดของกฎ additivity ซึ่งเป็นข้อบังคับในวิธีการของเบย์ กล่าวคือ เราสามารถเชื่อและไม่เชื่อความจริงหนึ่ง ๆ ได้ในเวลาเดียวกันถ้าเรามีหลักฐานทั้งประเภทที่สนับสนุนและแย้งกับความจริงที่เราตั้งเป็นสมมติฐาน ตัวอย่างของแนวทางนี้ได้แก่ Dempster-Shafer theory [SHAFER] หรือ certainty factor ของ Shortliffe [SHORTLIFFE]

ตรรกะแบบ nonmonotonic อีกแนวทางหนึ่งที่ใช้กลไก เป็นวิธีการที่พัฒนาไว้น้อยมาก วิธีเหล่านี้เป็นวิธีการในระบบทดลอง ขอยกตัวอย่างเช่น Truth maintenance System ของ [DOYLE] หรือ Theory of Endorsement ของ R. Cohen (Stanford) เป็นต้น ในที่นี้จะไม่ขออธิบายรายละเอียดของวิธีการเหล่านี้ แต่จะแสดงการใช้วิธีตามแนวทางของทฤษฎีความอาจเป็นไปได้ในโปรแกรมมัยซินเป็นตัวอย่าง (ตัวอย่างนี้ได้ทำให้ง่ายลงเพื่อให้พอจะเห็นวิธีการได้รวดเร็ว ดูรายละเอียดของกฎใน [SHORTLIFFE])

การอนุมานโดยใช้ certainty factor

การอนุมานแบบของโปรแกรมมัยซินใช้กฎ modus ponens ของตรรกะแบบมาตรฐานเป็นหลัก แต่มีการประเมินความจริงของหลักฐานและข้อสรุปทุกข้อ

ด้วยค่า certainty factor ซึ่งต่อไปจะย่อเป็น CF ดังนั้นจะอธิบายการใช้ modus ponens ด้วยตัวอย่างง่าย ๆ ให้ดูก่อนแล้วจึงอธิบายเกี่ยวกับการประเมินความจริงในภายหลัง

สมมติว่าเรามีความรู้สองข้อดังนี้

ข้อที่ 1 IF a THEN b
ข้อที่ 2 IF b THEN c

หากมีข้อเท็จจริงหรือหลักฐานว่า a เป็นความจริง เราจะสามารถทำการอนุมานได้ในสองลักษณะคือ โดยการอนุมานแบบ "โยงไปหน้า" (forward chaining) หรือแบบ "โยงถอยหลัง" (backward chaining) ทั้งสองแบบจะแสดงได้ดังรูปที่ 5

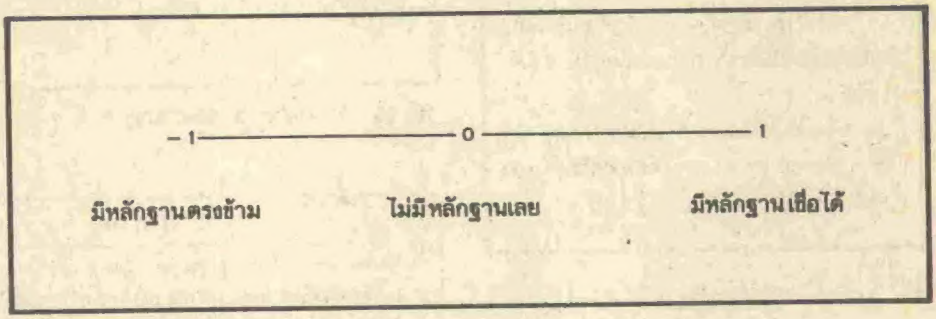
แบบโยงไปหน้า		แบบโยงถอยหลัง	
กฎข้อที่ 1	IF a THEN b	(เป้าหมาย)	ให้พิสูจน์ c
กฎข้อที่ 2	IF b THEN c	(กฎข้อที่ 2)	IF b THEN c
หลักฐาน	a.....	(กฎข้อที่ 1)	IF a THEN b
สรุป	c	(พยายามหา)	a
		(หลักฐาน)	a.....
		(สรุป)	c

รูปที่ 5 การอนุมานแบบโยงไปหน้าและแบบโยงถอยหลัง

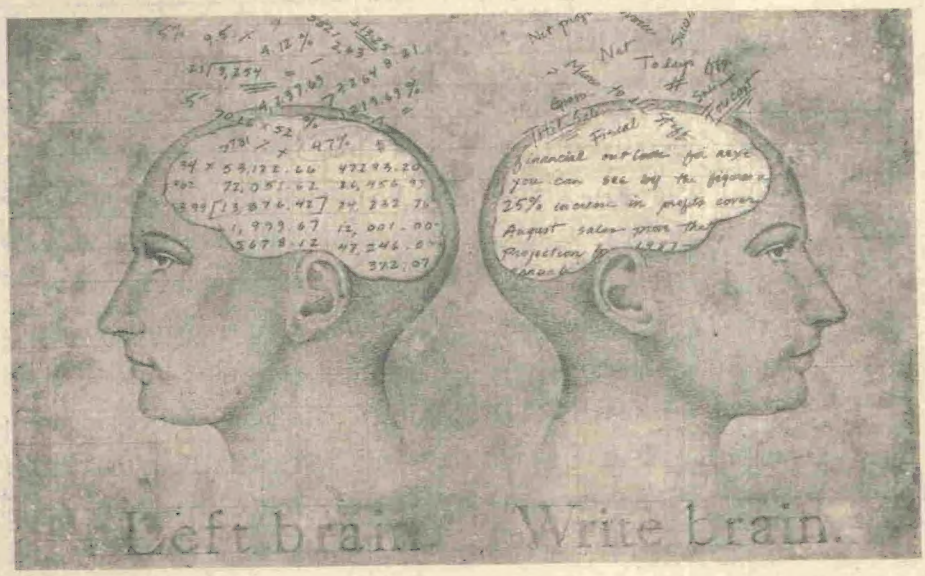
ข้อควรสังเกตในการทำงานแบบ "โยงไปหน้า" เราต้องเริ่มจากการมีหลักฐานอยู่ก่อน ดังนั้นโปรแกรมที่แก้ปัญหา โดยวิธีที่ผู้ใช้ป้อนข้อมูลที่มีอยู่ให้ทั้งหมดในคราวเดียว จึงสะดวกที่จะเขียนโดยการใช้กลไกควบคุมการอนุมานแบบนี้ ส่วนโปรแกรมที่รวบรวมหลักฐานทีละอย่างจากผู้ใช้จะเขียนได้ง่ายโดยการใช้กลไกแบบ "โยงถอยหลัง"

IF 1) The stain of the organism is gram positive, and
2) The morphology of the organism is coccus, and
3) The growth conformation of the organism is chains
THEN : There is suggestive evidence
(.7) That the identity of the organism is streptococcus

รูปที่ 6 กฎที่เป็นความรู้ของมัยซิน



รูปที่ 7 ค่าของความเป็นไปได้



มัยซินทำงานโดยวิธีหลังนี้ แต่กฎทุกข้อที่เป็นความรู้ของมัยซินมีค่า CF อยู่ด้วย ตัวอย่างดังรูปที่ 6

ค่า .7 (อยู่ในส่วน consequent) ได้แก่ ค่าของความเป็นไปได้ของหลักเกณฑ์ข้อนี้ คือค่า CF นั่นเอง ค่านี้จะอยู่ระหว่าง −1 ถึง 1 กฎข้อนี้กล่าวว่า "ถ้าเงื่อนไขต่อไปนี้...เป็นจริงมีหลักฐานให้เชื่อได้ 70% ว่าชนิดของเชื้อเป็น streptococcus"

ค่า CF นี้ เป็นค่าที่ขึ้นกับหลักฐาน คือมีความหมายดังแสดงเป็นเส้นมาตราดังรูปที่ 7

สมมติฐานแต่ละข้อมีค่าความเชื่อ (measure of belief ย่อว่า MB) และค่าความไม่เชื่อ (measure of disbelief ย่อว่า MD) อยู่คู่หนึ่ง ซึ่งในการอนุมานใช้ความรู้แต่ละข้อมีการคำนวณเพื่อปรับค่าคู่นี้ จนในที่สุดเมื่อเสร็จสิ้นการอนุมานค่า CF ของสมมติฐานข้อนั้นก็จะหาได้จาก CF = MB − MD ซึ่งผลที่ได้นี้จะนำมาเทียบเคียงกันระหว่างสมมติฐานที่ต่างกัน ทำให้โปรแกรมสามารถเลือกข้อสมมติฐานที่มีค่า CF สูงสุดเป็นคำตอบของปัญหาได้

ค่าความเชื่อและค่าความไม่เชื่อในมัยซินมีนิยามดังรูปที่ 8

MB [h, e] = x คือค่าความเชื่อเพิ่มขึ้นของสมมติฐาน h จากหลักฐาน e มีค่า x

MD [h, e] = y คือค่าความไม่เชื่อเพิ่มขึ้นของสมมติฐาน h จากหลักฐาน e มีค่าเท่ากับ y

โดยมีเงื่อนไขว่า 0 ≤ MB [h, e] ≤ 1 และ 0 ≤ MB [h, e] ≤ 1 โดยค่าเริ่มต้นของทั้งคู่เป็น 0

รูปที่ 8

สมมติว่าในการพิสูจน์สมมติฐานว่าเชื้อเป็นชนิดสเตรปโตค็อกคัสใช่หรือไม่ และเรามีความรู้เพียงข้อเดียวดังตัวอย่างข้างต้น โปรแกรมอาจถามหาหลักฐานจากผู้ใช้จนได้คำตอบดังตารางที่ 1

เราจะใช้การเขียน CF [X : Y] = Z เพื่อหมายความว่า CF ของ X เมื่อมีหลักฐาน Y เท่ากับ Z และเราจะใช้ e แทน "หลักฐานเท่าที่แล้ว" จะได้ดังรูปที่ 9

เราจะเรียกเงื่อนไขแต่ละข้อในกฎของตัวอย่างข้างต้นเป็น s1, s2 และ s3 ตามลำดับ และคำนวณ CF ของสมมติฐานข้อนี้จะได้ดังรูปที่ 10

สรุปได้ว่า "มีหลักฐานให้เชื่อได้ 42% ว่า ชนิดของเชื้อเป็น STREPTOCOCCUS"

การใช้ certainty factor เป็นวิธีการที่นิยมพอสมควรเนื่องจากเป็นวิธีที่ทราบกันแพร่หลาย อย่างไรก็ตามวิธีนี้ไม่มีพื้นฐานทางคณิตศาสตร์ที่ดี จึงเกิดความพยายามหลายกระแสที่ต้องการจะเสนอวิธีการอื่น Dempster-Shafer theory ซึ่งใช้ฟังก์ชันความเชื่อ (belief function) เป็นการเลียนแบบฟังก์ชันความเป็นไปได้ แต่ละเงื่อนไขของ additivity เสีย วิธีการนี้มีข้อดีที่สามารถรวมเอาวิธีการของเบย์ไว้เป็นกรณีพิเศษ ซึ่งทำให้น่าเชื่อถือกว่าวิธีการในมัยซิน แต่อย่างที่ได้กล่าวแล้วว่าการพิสูจน์ความถูกต้องของโปรแกรมระบบผู้เชี่ยวชาญทำได้โดยการใช้งานจริง ๆ เท่านั้น และโปรแกรมมัยซินก็ได้พิสูจน์ตัวเองจนประสบผลสำเร็จแล้ว

```
CF [POSITIVE GRAM STAIN : e]   = 1 เพราะคำตอบแน่นอน
CF [MORPHOLOGY COCCUS : e]     = 1 เพราะคำตอบแน่นอน
CF [GROW IN CHAINS : e]        = MB [GROW IN CHAINS : e]
                                 − MD [GROW IN CHAINS : e]
                               = 0.6 − 0
                               = 0.6
```

รูปที่ 9 ค่าของ CF

```
CF [STREPTOCOCCUS : s1 & s2 & s3]
        = MB [STREPTOCOCCUS : s1 & s2 & s3]
          − MD [STREPTOCOCCUS : s1 & s2 & s3]
        = CF ของกฎ × max (CF [s1 & s2 & s3 : e]) − 0
โดยที่ CF [s1 & s2 & s3 : e] = min (MB (s1 : e), MB (s2 : e), MB (s3 : e))
                             = min (1, 1, 0.6)
                             = 0.6
เพราะฉะนั้น CF [STREPTOCOCCUS : s1 & s2 & s3] = 0.7 × 0.6 − 0
                                               = 0.42
```

รูปที่ 10 การคำนวณ CF ของสมมติฐาน

คำถาม	คำตอบ
Is the stain of the organism gram positive or negative?	positive
Is the morphology of the organism coccus?	Yes
Did the organism grow in clumps, chain, or pairs?	CHAINS (.6) PAIRS (.3) CLUMPS (−8)

ตารางที่ 1 โปรแกรมถามหาหลักฐานจากผู้ใช้

สรุป

บทความฉบับนี้อธิบายสถาปัตยกรรมของส่วนแกนของโปรแกรมระบบผู้เชี่ยวชาญในแบบ rule-based ซึ่งเป็นสถาปัตยกรรมที่นิยมที่สุดในปัจจุบัน ตั้งแต่โมเดลที่เป็นต้นกำเนิดของความคิด จนถึงการทำงานของกลไกระดับล่าง โดยพยายามเน้นความเข้าใจถึงความคิดที่เป็นหัวใจของกระบวนการทำงาน

ปัญหาที่สำคัญของโปรแกรมระบบผู้เชี่ยวชาญที่เป็นจุดสนใจในปัจจุบันมีอยู่เป็นจำนวนมาก เช่น

1. สร้างโมเดลการอนุมานที่ดีกว่าโมเดลในปัจจุบัน

2. การจัดการกับความรู้ที่กำกวม เช่น เงื่อนไขในกฎที่อาจจะกล่าวว่า "สีของสารละลายค่อนข้างแดง" จะทราบได้อย่างไรว่า คำว่า "ค่อนข้าง" จะเป็นจริงเมื่อไร

3. การหาวิธีการพิสูจน์ความเป็นอิสระต่อกันของความรู้แต่ละข้อ

4. พิสูจน์ completeness และ consistency ในทางปฏิบัติของระบบ (การพิสูจน์ในทางตรรกศาสตร์ทำไม่ได้ แต่การตรวจโดยวิธีอัตโนมัติเพื่อหาความรู้ข้อที่หายไป ทำให้การอนุมานไม่สามารถบรรลุได้ในบางสาย หรือความรู้ที่ขัดแย้งกันเองเหล่านี้สามารถทำได้ แต่วิธีการมาตรฐานเหล่านี้ยังไม่มีในปัจจุบัน วิธีการทางทฤษฎีเหล่านี้จะมีประโยชน์ในการสร้าง expert system editor)

5. วิธีการทดสอบระบบ

6. การสร้างระบบแปลความรู้เป็นคำสั่งโดยอัตโนมัติ

7. การสร้างโปรแกรมอัตโนมัติที่สามารถตอบคำถามผู้ใช้โดยการอนุมานจาก meta knowledge เป็นการใช้เหตุผลเพื่ออธิบายการใช้เหตุผลของระบบ

8. การสร้างระบบผู้เชี่ยวชาญเป็นส่วนหลัง (backend) ของระบบอื่น ๆ เช่น ระบบการจัดการฐานข้อมูล (DBMS) ระบบการประมวลเอกสาร (document processing) ฯลฯ

ในระยะหลังสถาปัตยกรรมของส่วน support systems สำหรับระบบผู้เชี่ยวชาญได้รับความสนใจเป็นพิเศษ เนื่องจากประสบการณ์การสร้างซอฟต์แวร์ประเภทนี้ได้แสดงให้เห็นว่า support systems มีส่วนโดยตรงกับการยอมรับระบบของผู้ใช้ และมีทางที่จะสร้างโปรแกรมมาตรฐานทำนองเดียวกับเซลล์ได้ ในไม่ช้าอาจจะมีการสร้างเซลล์สำหรับไมโครคอมพิวเตอร์ที่มีส่วนของ support system ดี ๆ ออกมาในท้องตลาด ทำให้การสร้างโปรแกรมระบบผู้เชี่ยวชาญง่ายขึ้นกว่าในปัจจุบันและมีวิธีปฏิบัติที่เป็นมาตรฐานมากขึ้นอีก

หนังสืออ้างอิง

BROWNSTON, LEE et. al. Programming Expert Systems in OPS5 : An Introduction to Rule-Based Programming Addison-Wesley, 1985.

COLLINS, A., & QUILLIAN, M.R. Retrieval Time from Semantic Memory Journal of Verbal Learning & Verbal Behavior, No.8, 1969.

DAVIS, RANDALL & KING, JONATHAN The Origin of Rule-Based Systems in AI in Rule-Based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project edited by Bruce G. Buchanan and Edward H. Shortliffe, Addison-Wesley, 1984.

DOYLE, J. A Truth Maintenance System Artificial Intelligence, vol 12, No.3, 1979.

FAIRLEY, RICHARD Software Engineering Concepts McGraw-Hill, 1985.

HARMON, PAUL & KING, DAVID Expert Systems: Artificial Intelligence in Business John Wiley & Sons, 1985.

RICH, ELAINE Artificial Intelligence McGraw-Hill, 1983.

SHAFER, GLENN A Mathematical Theory of Evidence Princeton University Press, 1976.

SHORTLIFFF, EDWARD E. & BUCHANAN, BRUCE G. A Model of Inexact Reasoning in Medicin Mathematical Biosciences, 23, 1975.

YU, VICTOR L. et. al. An Evalustion of MYCIN's Advice in Rule-Based Expert Systems : The MYCIN Experiments of the Standford Heuristic Programming Project edited by Bruce G. Buchanan and Edward H. Shortliffe, Addison-Wesley, 1984.